In the specification: (as published June 3 '04)

Kindly amend the specification at paragraphs 16 and 17 as follows:

Accordingly, it is a principal object of the present invention to overcome the limitations of prior art and provide a method and a system that ~~substantially~~ replaces the writing of source code for developing software applications, and to achieve real visualization of the solution itself.

It is still a further object of the present invention that instead of writing code, a model is created by the developer, wherein the model includes ~~substantially~~ everything needed for defining the application in adequately precise terms.

Kindly amend the specification at paragraph 118 as follows:

After receiving its two mandatory trigger inputs, the action of Concat 320 is initiated. The result of Concat 320 then flows through Out exit 330 and is sent 335 to the exit output 340 of Handle_Header 300. The data model that is exposed through Out exit 330 of Concat 320 and the data model that is exposed through the output exit 340 of Handle_Header 300 should both be of the same type.

Kindly amend the specification at paragraph 119 as follows:

Fig. 4 is an exemplary schematic illustration of the Handle Execution 400 data manipulation process model, a composite action model, which starts with handling a received message, constructed in accordance with the principles of the present invention. Handle Execution 400 receives a string buffer In 410 containing the received message, and sends it 411 "as is" to a parsing action called Parse Message 420. Parse Message 420 receives a string buffer 'In 2' 422 (the text to be parsed) and a data model 'In 1' 421 (the text's format definition), and executes the parse plug-in which parses the string buffer 422 based on the formatting information of the data model 421. The

output 423 of Parse Message 420 is sent out 425 as an "Extract Details" 430 [a] data tree message 432 containing the parsed data 431 (a composite tree structure).

Kindly amend the specification at paragraph 123 as follows:

The composite process model Parse Full Message 500 has one trigger, 'In 1' 505, from which it receives a string buffer that is sent 506 to Parse Incoming Msg 510, which receives the buffer as Data 511 along with a data model Model 512 describing the structure of that buffer, and executes the Parse plug-in. The result of the parse atomic action 513, containing the parsed data, is transferred 515 to Extract MT action 540 as Parsed Data 541. Extract MT composite action 540 receives Parsed Data 541 and extracts pieces of information from it, i.e. the MT field content exposed through exit MT 542 and the text block field content exposed through exit Text Block 543, which are both transferred 545, 546 to Switch action 520.

Kindly amend the specification at paragraph 129 as follows:

Handle Allocation 700 is a composite action model that receives an input data model In 710, which contains a repetition of Allocations 725. Allocations 725 are transferred, one by one, to instances of the Extract Allocation Details 730, which is a repetitive action. The flow 726 from Allocations 720 to the trigger 731 of the action Extract Allocation Details 730 occurs many times, according to the number of instances of Allocations 725. Each time the flow occurs, an instance of Allocations data model 725 is transferred to a new instance of Extract Allocation Details 730. A new instance of Allocation Data 732 is then created per each instance of Extract Allocation Details 730. This newly created instance of Allocation Data 732 is sent 735 through exit 733 of the current instance of Extract Allocation Details 730 to exit 734 of the parent process 730, thus the instance of Extract Allocation Details 730 finishes. The

repetitive nature of Extract Allocation Details 730 is represented in the diagram by multiple rectangles overlapping each other.

Kindly amend the specification at paragraph 132 as follows:

The system 'Simple System' 800 of Fig. 8 is composed of only input and output adapters. I.e., it only receives a message through the Input Adapter sub-process 810 and sends it 815 out through the Output Adapter sub-process 820. Input Adapter 810 and Output Adapter 820 are atomic systems. Messages sent 815 from Input Adapter 810 are received by Output Adapter 820 through the asynchronous input slot In 821. Unlike triggers, asynchronous input can receive data while a process is alive and running.

Kindly amend the specification at paragraph 134 as follows:

A main process is a process which is a direct child of a system. Main processes must be repetitive because they receive an unpredictable amount of messages as time passes. Each message triggers an instance of the main process and is processed by it. In FIG. 9, each message coming from Input Adapter 910 invokes 915 a new instance of process Handle Message 930. Each instance of Handle Message 930 exposes a new output, which is transferred 925 to Output Adapter 920. This example does not contain exception handling, therefore if an exception occurs, for example, an incoming message fails parsing, then the runtime engine's default exception handling takes place.

Kindly amend the specification at paragraphs 135 and 136 as follows:

Fig. 10 is an exemplary schematic illustration of a typical asynchronous process model Trade 1000, composed of a main data element 'Main Data' 1005, comprising Trade Details 1008, and three sub-processes, constructed in accordance with the principles of the present invention. The three sub-processes

are Initialize 1010, which initializes Trade Details 1008, Confirm 1020, which prepares confirmation details to confirm that the execution message has been received 1015, and Update Status 1030, which updates the trade status in various phases during the lifecycle of the trade.

Update Status 1030 updates the status of the trade only after a confirmation message is received 1025. Since a confirmation is likely to be received after Trade 1000 is started, an asynchronous input slot is preferably used for that purpose. An asynchronous input slot, e.g. slot 1040, is the only kind of slot that enables a process to receive data after it has started. Trade 1000 finishes after Update Status 1030 is finished.

Kindly amend the specification at paragraph 142 as follows:

The way Confirmation data 1118 is distributed 1127 is through the asynchronous input slot 1124 of Trade 1120. Input slot 1124 has an Addressing Clause attached to it, used as a matching condition (or distribution condition). Each instance of the output 1118 is distributed to a matching instance of Trade 1120, selected according to the addressing clause. The simplest distribution condition compares one of the fields of the data object received through input slot 1124 to the corresponding field in the main data 1121 of Trade 1120. If they match, the instance of Confirmation data 1118 goes through input slot 1124 and enters the matching instance of Trade 1120. As a result, the flow from input slot 1124 is executed and triggers the Update Status action 1123, which updates the trade status to indicate that a Confirmation Message 1118 has been received.

Kindly amend the specification at paragraph 145 as follows:

Fig. 12 is an exemplary schematic illustration of a complete system model System 1200, constructed in accordance with the principles of the present invention. When an execution

message arrives 1215 at trigger 1221 through Input Adapter 1210, a new instance of the Switch process 1220 is created. The message exits through the Exec exit 1223 of Switch process 1220, and flows 1225 through the In trigger 1231 of 'Handle Exec.' process 1230. A new instance of 'Handle Exec.' 1230 is created. 'Handle Exec.' process 1230 extracts the relevant data from the message and sends it via the Flow 1235 to a new trade instance of the Trade process 1240. Then a trade confirmation message is created by the Generate Confirmation process 1250, and it is sent 1255 out via Output Adapter 1260.

Kindly amend the specification at paragraph 147 as follows:

Fig. 13 is an exemplary schematic illustration of exception handling within a process model 1300, herein achieved by modeling a special fail exit 1315, and defining the flow from it, constructed in accordance with the principles of the present invention. Fail exit 1315 of the Divide action 1310 catches exceptions of pre-defined types, e.g. division by 0, which are considered as business exceptions, and thereafter the modeler is free to model the required exception behavior. Fail exit 1315 triggers 1318 a Report Error action 1320. If a process does not have a fail exit, and an exception occurs, the exception floats up to the nearest fail exit. If no fail exit is found, the runtime engine catches the exception and executes a default exception handling.